<System On Chip>

- PS2 keyboard Transmitter & receiver-

· 제출 레포트

전자공학과

2015142011 류찬

목 차

Personal System/2의 구현

➔ Personal System/2 : IBM PC의 Keyboard와 마우스의 입력 인터페이스이다.

Keyboard의 구현 방식

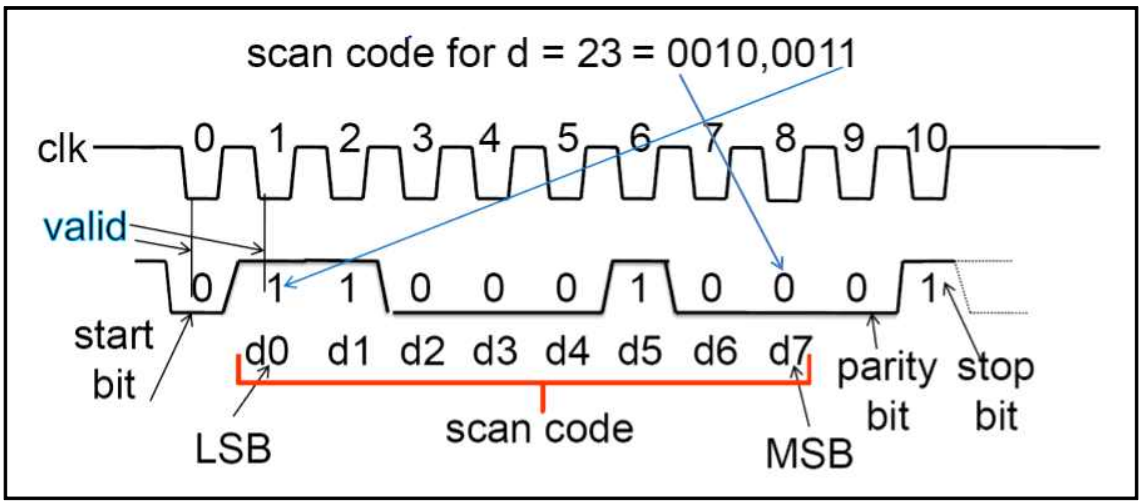
+ 키가 눌렸을 때 : 11-bit 하나의 code 전송
+ 키가 떨어질 때 : 두 개의 SCAN Code 전송(F0 -> Key code)
+ 키가 계속 눌려있을 때 : 해당 Key Code를 100ms마다 전송

For example

When input ‘d’,

Scan code for ‘d’ = 23 = 0010,0011

즉 클럭과 데이터 파형은

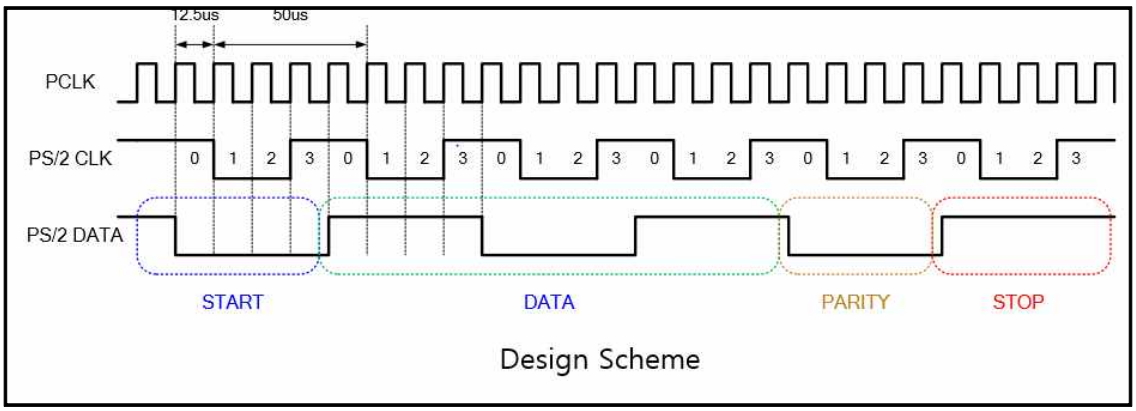


다음과 같이 나타난다.

위와 같은 방식으로 Keyboard의 Transmitter와 Receiver을 구현할 예정이다.(이번 학기의 실습에는 Keyboard의 데이터 입력만 처리한다.)

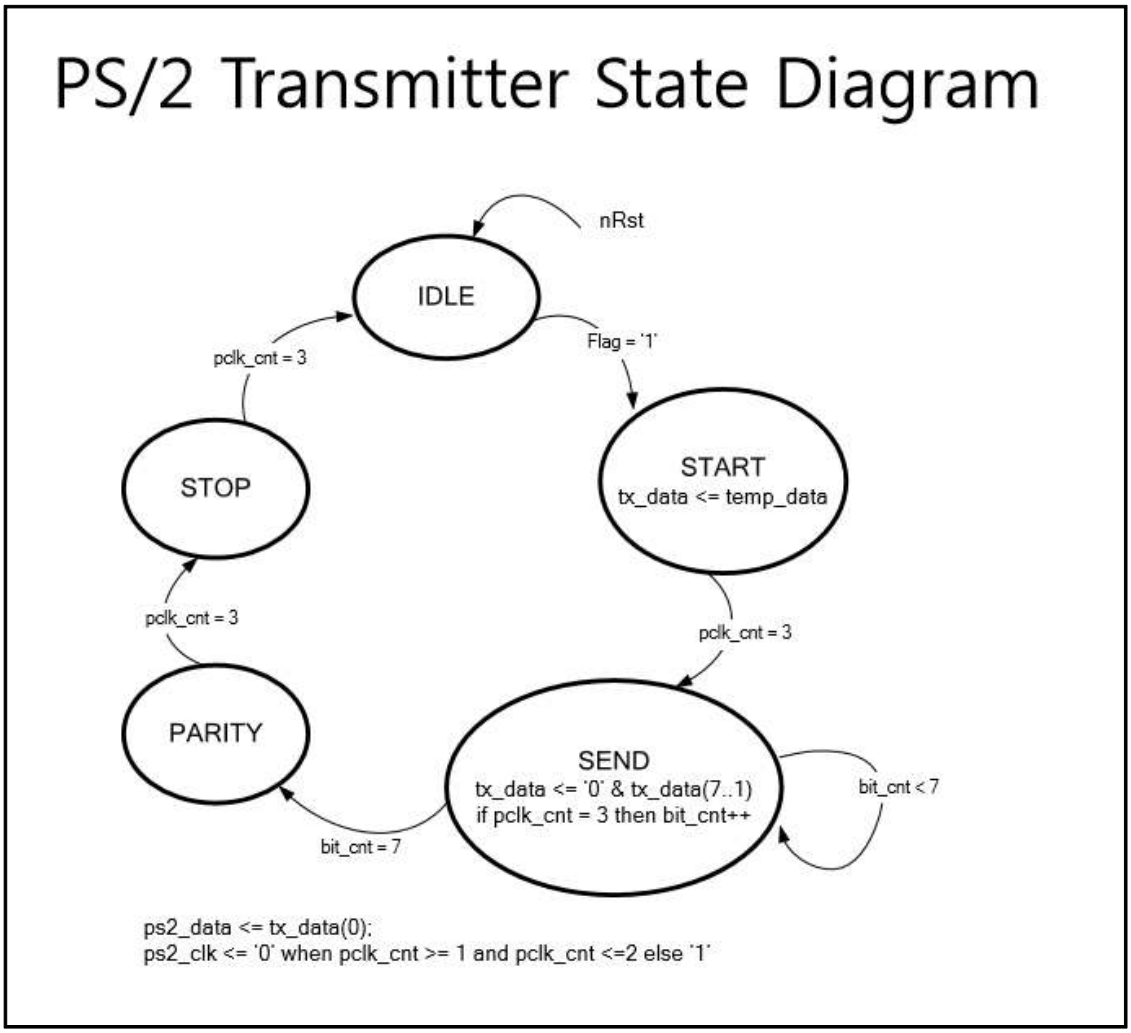
시뮬레이션을 위한 PS/2의 데이터 Transmitter의 설계 조건이다.

- System CLK : 100MHz
- PS/2 CLK : 20KHz (50us)
- PS/2 PCLK : 80KHz



Design Scheme

위의 Design Scheme을 통해 보면
처음 PS/2 DATA가 falling edge일 때 Transmitter는 Start된다.
PS/2 CLK에서 data전송은 High-Low-Low-High(1,0,0,1)가 한 주기로 사용된다.
PS/2 PCLK는 PS/2 CLK의 4 배에 해당한다. (즉 주기의 길이는 1/4이라 할 수 있다.)
따라서 4개의 PS/2 CLK를 한 세트로 DATA를 동기화시켜 전송한다.



State Diagram을 통해 알 수 있는 사실로는

1) nRst이 '0'인 상태일 때에, State는 IDLE인 상태이다. (즉, 모든 동작은 unable되어 있다.)
2) Flag가 '1'인 상태에서 IDLE상태에서 START상태로 이동한다.

➔ START 상태에서는 tx_data에 temp_data의 값을 저장한다.

3) pclk_cnt가 3이 되면 START상태에서 SEND상태로 이동한다.

➔ SEND 상태에서는 tx_data에 (0과 tx_data의 7부터 1까지의 비트를 'and')하여 저장한다.

➔ 이후 pclk_cnt의 값이 3이면 bit_cnt의 값에 1을 더한다.

➔ 이를 bit_cnt가 7이 될 때 까지 반복한다.

4) bit_cnt가 7이 되었으면 SEND 상태에서 PARITY 상태로 이동한다.

➔ PARITY비트는 ODD parity이다. (즉, '0'으로 고정된다.)

5) bit_cnt가 5가 되면 STOP 상태로 이동한다.

➔ 여기서 bit출력은 항상 1이 된다.

6) pclk_cnt가 3가 되면 STOP 상태에서 IDLE 상태로 이동한다.

다음으로 PS/2 Transmitter를 코딩한다.

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_arith.all;
4   use ieee.std_logic_unsigned.all;
5
6   entity ps2_keyboard is
7     port
8     (
9       nRst      : in std_logic;
10      clk       : in std_logic;   -- 100MHz, 10ns
11      start_sig : in std_logic;
12      data      : in std_logic_vector(7 downto 0);
13      ps2_clk   : out std_logic;
14      ps2_data  : out std_logic
15    );
16
17  end ps2_keyboard;
```

코딩에 필요한 library를 선언하고 entity를 만든다.
entity는 PS/2 Transmitter의 구조를 의미한다. 즉 nRst, clk, start_sig, data은 입력 포트로 사용되고, ps2_clk와 ps2_data는 출력 포트로 사용된다.
추가로 data는 std_logic_vector(7 downto 0)로 선언되어 있는데, 이는 다른 포트들(1bit)과는 다르게 data 포트는 7부터 0의 8개 bit를 사용한다는 것을 알 수 있다.

```
19  architecture BEH of ps2_keyboard is
20
21    type state_type is (IDLE, START, SEND, PARITY, STOP);
22      signal state        : state_type;
23      signal cnt          : std_logic_vector(9 downto 0); -- 1024
24      signal pclk         : std_logic;
25      signal pclk_cnt     : std_logic_vector(1 downto 0);
26      signal bit_cnt      : std_logic_vector(7 downto 0);
27      signal temp_data    : std_logic_vector(7 downto 0);
28      signal tx_data       : std_logic_vector(7 downto 0);
29
30      signal start_d      : std_logic;
31      signal flag         : std_logic;
```

다음으로 architecture 부분을 설정한다.
Architecture를 통해 우리는 PS/2 Transmitter의 실행을 유도할 수 있다. 즉 architecture문은 Transmitter의 행동(BEHave)를 설정 가능하다.
먼저 type문을 통해 state_type을 IDLE, START, SEND, PARITY, STOP의 5가지 상태에 관해 설정한다.
이렇게 설정한 state는 state_type의 signal로 설정한다.

다음으로 각 signal을 설정한다.
이렇게 signal로 설정된 각 데이터는 entity 내부에서 데이터를 주고받을 수 있게 한다.

```
35      begin
36
37        process(nRst, clk)
38          begin
39            if(nRst = '0') then
40              cnt     <= (others => '0');
41              pclk    <= '0';
42            elsif rising_edge(clk) then
43              if(cnt = 624) then
44                cnt     <= (others => '0');
45                pclk    <= not pclk;
46              else
47                cnt     <= cnt + 1;
48              end if;
49            end if;
50        end process;
```

begin문 이후로는 architecture의 실제 서술부가 시작된다.
이는 process에서 실행된다.

process문은 괄호 안의 nRst, clk의 값에 따라 순차적으로 진행된다.
먼저 nRst이 '0'일 때 (40~41줄의 내용을 실행)
cnt, pclk의 값에 '0'을 넣어 초기화한다. (여기서 cnt는 vector값으로 다양한 비트를 갖는데, 그 모든 값을 '0'으로 초기화했다.)

그리고 nRst의 값이 만약 '0'이 아니라면(여기서는 bit를 사용하므로 '1'일 때를 의미한다)
그 상황에서 클럭의 rising_edge(상승 엣지)가 일어나면, cnt의 상태에 따라 진행하는데
이전 상태에서 cnt의 모든 값은 '0'으로 초기화 되어서 43줄에 앞서 46줄이 진행될 것이다.

46줄에서 cnt는 cnt+1의 값이 저장되게 된다.

이후 cnt의 값이 624에 도달하면 cnt를 "0000000000"으로 초기화 시킨 뒤 pclk의 값을 not을 통해 반전시킨다(~pclk)

➔ (0~624)x2를 통해 1250번 마다 한 클럭을 생성할 수 있게 한다.

```
53      process(nRst, clk)
54        begin
55          if(nRst = '0') then
56             start_d   <= '0';
57             flag      <= '0';
58             temp_data <= (others => '0');
59          elsif rising_edge(clk) then
60            start_d    <= start_sig;
61            if(start_d = '0') and (start_sig = '1') then
62             flag      <= '1';
63             temp_data <= data;
64            elsif(state = START) then
65             flag      <= '0';
66           end if;
67         end if;
68       end process;
```

이번 process에서는 nRst, clk의 값에 따라 start_d, flag, temp_data에 변동을 주게 된다.
먼저 nRst가 '0'인 상태에서
start_d와 flag, temp_data를 '0'으로 초기화시킨다.

이후 clk가 rising edge상태일 때에는 다른 조건과 관계 없이 start_d에 start_sig의 값을 저장한다.

1) 이후 start_d의 값은 '0'이고, start_sig의 값이 '1'이라는 조건이 충족된 상태에서 flag에는 '1'을 대입하고, temp_data에는 data의 값을 저장한다.
2) (1)의 조건이 충족되지 않은 상태에서 state의 값과 START의 값이 같다면 Flag를 다시 '0'으로 초기화시킨다.

```
71      process(nRst, pclk)
72        begin
73          if(nRst = '0') then
74            state     <= IDLE;
75            pclk_cnt  <= (others => '0');
76            bit_cnt   <= (others => '0');
77            tx_data   <= (others => '0');
```

nRst의 값이 '0'일 때 state는 IDLE상태가 된다.
이후 pclk_cnt와 bit_cnt, tx_data의 세 부분을 모두 초기화한다.

```
78          elsif rising_edge(pclk) then
79            case state is
```

pclk가 rising edge인 상태에서 state의 값에 따라 각각의 상태를 진행시킨다.

```
81              when IDLE =>
82                if(flag = '1') then
83                  state <= START;
84                else
85                  state <= IDLE;
86                end if;
87                  pclk_cnt  <= (others => '0');
88                  bit_cnt   <= (others => '0');
89                  tx_data   <= (others => '0');
```

먼저 state가 IDLE상태일 때(위의 프로세서가 진행된 상태라면 state는 IDLE상태에 해당될 것이다)
flag비트가 먼저 '1'로 세트되어 있는 상태라면 state를 START로 이동시킨다.
만약 flag비트가 '1'로 되어있지 않은 상태라면 다시 state의 값은 IDLE로 되돌아간다.
상태를 if문에 따라 진행한 뒤 pclk_cnt, bit_cnt, tx_data의 모든 값들을 '0'으로 초기화시킨다.

```
91              when START =>
92                if(pclk_cnt = 3) then
93                  pclk_cnt  <= (others => '0');
94                  state     <= SEND;
95                else
96                  pclk_cnt  <= pclk_cnt + 1;
97                  state     <= START;
98                end if;
99                  tx_data   <= temp_data;
```

다음으로 state가 START상태일 때(이전의 case에서 flag가 '1'이면 START의 값이 되었을 것이다.)
pclk_cnt의 값이 3일 때 다시 pclk_cnt의 값을 초기화 시킨 후 state에SEND를 저장하여 이후 단계로 진행하게 한다.
pclk_cnt의 값이 3이 아니라면, pclk_cnt의 값에 1을 더한 값을 저장시키고, state는 시작점인 START로 되돌아간다.
<즉, pclk_cnt의 값이 3이 될 때까지 pclk_cnt의 값을 증가시키고, 원하는 상태에 도달하면 pclk_cnt를 초기화 한 뒤 SEND상태로 진행한다.>

```
101             when SEND =>
102               if(pclk_cnt = 3) then
103                 pclk_cnt  <= (others => '0');
104                 if(bit_cnt = 7) then
105                   bit_cnt <= (others => '0');
106                   state   <= PARITY;
107                 else
108                   tx_data <= '0' & tx_data(7 downto 1);
109                   bit_cnt <= bit_cnt + 1;
110                   state   <= SEND;
111                 end if;
112               else
113                 pclk_cnt  <= pclk_cnt + 1;
114                 state     <= SEND;
115               end if;
```

State의 값이 SEND가 되었을 때
102줄과 112줄과 연결되어 있다.
pclk_cnt의 값이 3이 아니라면 state는 이 이상 진행되지 않고(state의 처음 when 도달값인 SEND를 다시 넣는다) pclk_cnt의 값을 1 추가한다.
이후 102줄 즉 pclk_cnt의 값이 3이 되었을 때 pclk_cnt를 모두 '0'으로 초기화한다.
또한 104줄과 107줄도 서로 연결되어 있다.
102줄이 실행될 때마다 bit_cnt의 값을 확인하여 그 값이 7이 아닐 때 bit_cnt에 1을 더한 값을 저장하고 tx_data에는 '0'과 tx_data(7 downto 1)을 서로 '&'즉, 연결연산 하여 저장시킨다.
-> 0이 왼쪽에 하나씩 추가되는 것으로 보아 오른쪽으로 shift연산을 진행함을 알 수 있다.
따라서 코드에서는 값이 절반씩 줄어드는 효과를 보일 것이라는 것을 짐작 할 수 있다.
이를 bit_cnt가 7이 될 때까지 반복한 후 bit_cnt가 7이 되었을 때 모든 bit_cnt의 값을 '0'd로 초기화 한 후 state에 PARITY를 넣어 진행시킨다.

```
117             when PARITY =>
118               if(pclk_cnt = 3) then
119                 pclk_cnt  <= (others => '0');
120                 state     <= STOP;
121               else
122                 pclk_cnt  <= pclk_cnt + 1;
123                 state     <= PARITY;
124               end if;
```

State의 값이 SEND가 되었을 때
pclk의 값이 3이 될때까지 값을 증가시키다 3이 되면 STOP으로 진행한다.

```
126                 when STOP =>
127                   if(pclk_cnt = 3) then
128                     pclk_cnt  <= (others => '0');
129                     state     <= IDLE;
130                   else
131                     pclk_cnt  <= pclk_cnt + 1;
132                     state     <= STOP;
133                   end if;
134
135                 when others =>
136                     state     <= STOP;
137                   end case;
138                 end if;
139               end process;
```

State가 STOP 상태가 되었을 때
pclk_cnt가 3이 될 때까지 증가시키다 3이 되면 IDLE상태로 넘어간다.

State가 위의 값들이 아닐 때에는 state는 STOP상태에 머문다.

```
142             ps2_clk  <= '0'          when pclk_cnt >=1 and pclk_cnt <= 2
143                                     else '1';
144
145             ps2_data <= tx_data(0)   when state = SEND else
146                         '0'          when state = START or state = PARITY
147                                     else '1';
148
149         end BEH;
```

pclk_cnt의 값이 1보다 크고 2보다 작을 경우(1 or 2일 경우) ps2_clk에 '0'을 할당한다.
그렇지 않을 경우(0 or 3일 경우) ps2_clk에 '1'을 할당한다.
State가 SEND일 때 ps2_data에 tx_data의 LSB 값을 할당한다.
State가 START이거나 PARITY일 때는 '0'으로 초기화한다.
그 외의 경우에 ps2_data의 값은 항상 '1'이 된다.

이후로 ps2_keyboard의 Testbench를 구현하여 시뮬레이션을 진행한다.

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_arith.all;
4   use ieee.std_logic_unsigned.all;
5
6   entity tb_ps2_keyboard is end;
7
8   architecture BEH of tb_ps2_keyboard is
9
10    component ps2_keyboard is
11      port
12      (
13        nRST            : in std_logic;
14        clk             : in std_logic;
15        start_sig       : in std_logic;
16        data            : in std_logic_vector(7 downto 0);
17        ps2_clk         : out std_logic;
18        ps2_data        : out std_logic
19      );
20    end component;
21
22    signal nRst         : std_logic;
23    signal clk          : std_logic;
24    signal start_sig    : std_logic;
25    signal data         : std_logic_vector(7 downto 0);
26    signal ps2_clk      : std_logic;
27    signal ps2_data     : std_logic;
28    signal internal_cnt : std_logic_vector(80 downto 0);
```

먼저 tb의 entity문은 단순히 tb_ps_keyboard를 구현시키는 것으로 끝낸다.

Component문에 실제로 내부에서 사용될 각 포트를 설정한다.

다음의 process를 진행하기 전에 signal을 통해 entity간의 동적 데이터를 주고받을 수 있게 각각의 요소를 설정한다.

```
31  begin
32
33    process
34      begin
35        if(NOW = 0ns) then
36          nRst  <= '0', '1' after 200ns;
37        end if;
38        wait for 1sec;
39      end process;
40
41      process
42        begin
43          clk <= '0', '1' after 5ns;
44          wait for 10ns;
45        end process;
46
47        process(nRst, clk)
48          begin
49            if(nRst = '0') then
50              internal_cnt <= (others => '0');
51            elsif rising_edge(clk) then
52              internal_cnt <= internal_cnt + 1;
53            end if;
54          end process;
```

세 개의 process문에 대해 설명하겠다.
먼저 nRst에는 '0'을 할당하다 200ns 뒤부터는 '1'을 넣는다.
이 '1'의 데이터는 1sec까지 유지된다.

다음 process에서 clk이 '0'이 들어온 후 5ns가 지나면 '1'을 10ns까지 유지시킨다.
즉 0, 1이 5ns로 반복되는 일종의 PWM신호라고 볼 수 있다.

마지막 process에서는 nRst이 '0' 일 때 internal_cnt의 모든 요소를 '0'으로 초기화한다.
이후 clk이 rising edge이고, nRst이 '1'인 상황에서 internal_cnt는 1씩 증가한다.

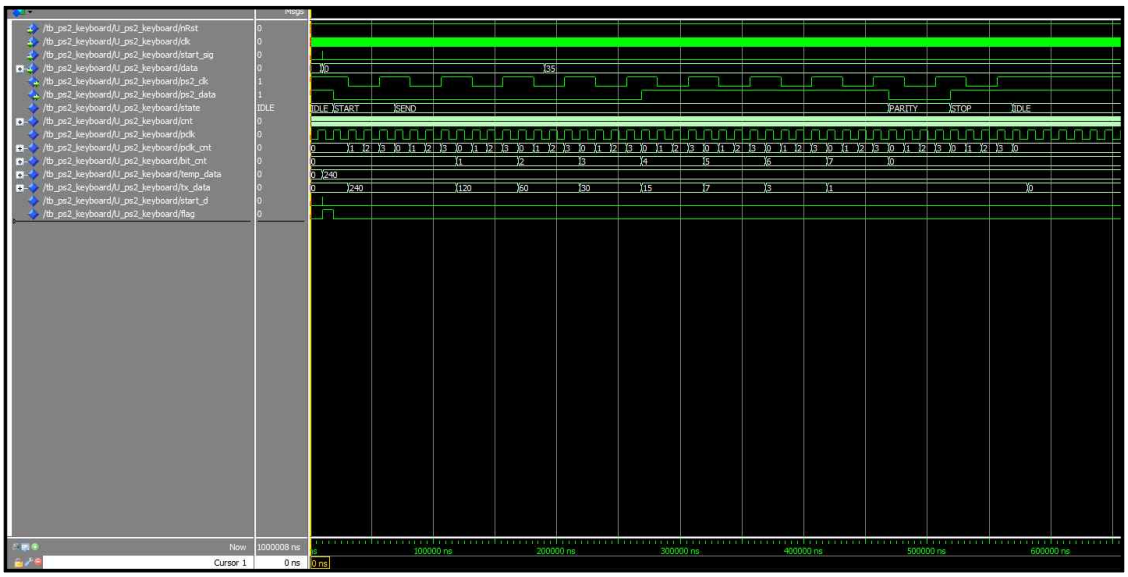
```
56          start_sig <= '1' when internal_cnt    =  1000 or internal_cnt  =  200000 else '0';
57          data      <= x"F0" when internal_cnt >=   900 and internal_cnt <=   1100 else
58                       x"23" when internal_cnt >= 19000 and internal_cnt <= 210000 else
59                                                (others => '0');
```

Internal_cnt의 값이 1000 or 200000일 때 start_sig의 값에 '1'을 할당한다.
-> 이외의 상황에서 start_sig는 '0'의 값을 갖는다.

Internal_cnt의 값이 900 ~ 1100 일 때 data 의 값에 0xf0을 할당한다.
Internal_cnt의 값이 19000 ~ 210000일 때 data 의 값에 0x23을 할당한다.
-> 이외의 상황에서 data 는 '0'의 값을 갖는다.

```
61          U_ps2_keyboard : ps2_keyboard
62          port map
63            (
64              nRst          => nRst,
65              clk           => clk,
66              start_sig     => start_sig,
67              data          => data,
68              ps2_clk       => ps2_clk,
69              ps2_data      => ps2_data
70            );
71
72          end BEH;
```

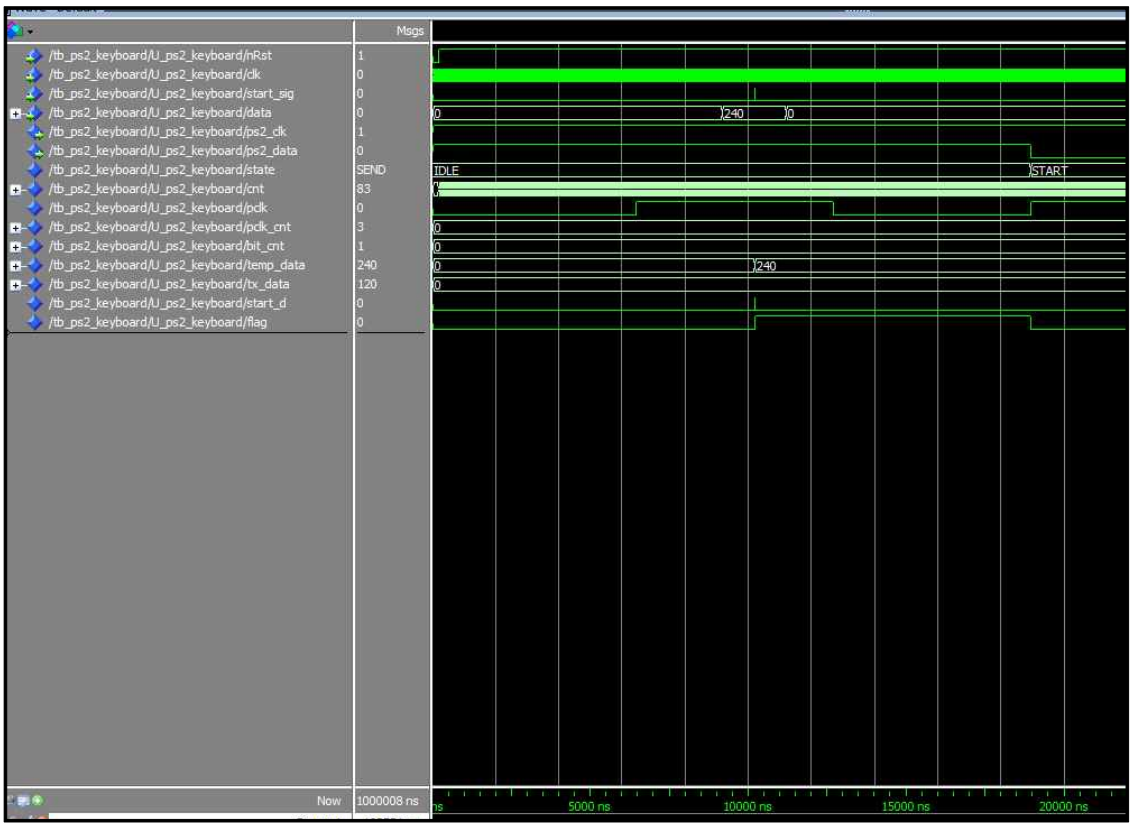
각 component에 port를 실제로 사용하기 위하여 입력한다.
Port map에서 연결됨을 알 수 있다.



다음은 위의 Test bench와 keyboard transmiiter을 구현하여 진행한 시뮬레이션의 결과이다

.

위 시뮬레이션에 관한 설명은 밑에서 한 장씩 하기로 하겠다.



먼저 IDLE상태에서 START로 진행하기 전까지의 상태이다.

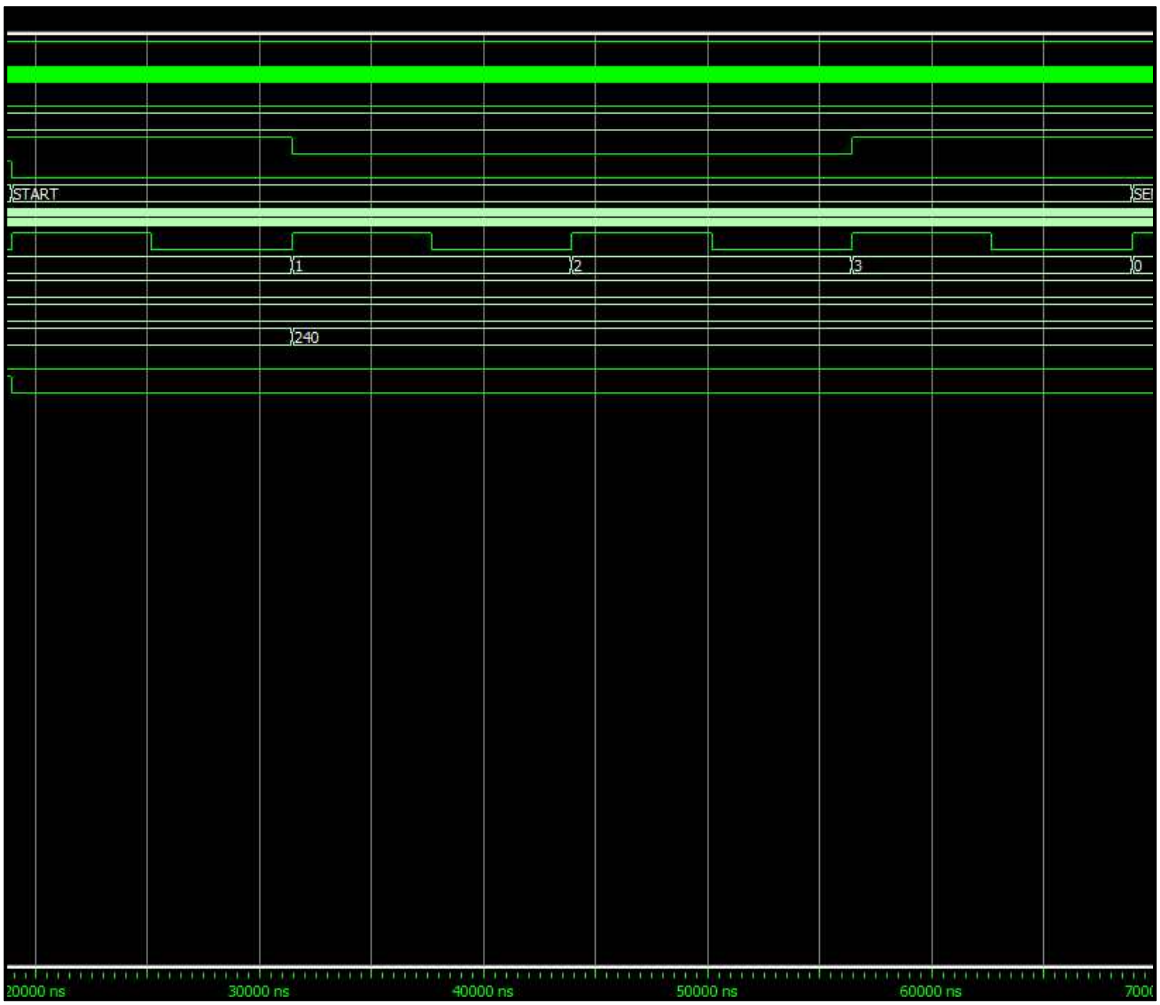
nRst 시그널은 '0'으로 존재하는데, 이 때 state는 IDLE상태이며 pclk_cnt, bit_cnt, tx_data는 모두 '0으로 초기화 된 상태이다.

nRst은 200ns 뒤에 '1'값을 부여받는다.

start_sig가 '1'인 상태가 되면 flag는 '1'이 된다. 그 때 temp_data에 data의 값이 입력된다.

pclk의 rising edge 상황에 맞추어 시뮬레이션은 판별하는데, 이 rising edge의 상황에서 flag가 '1'이면 state는 START로 이동한다.

이후, pclk_cnt, bit_cnt, tx_data는 모두 '0'으로 다시 초기화된다.

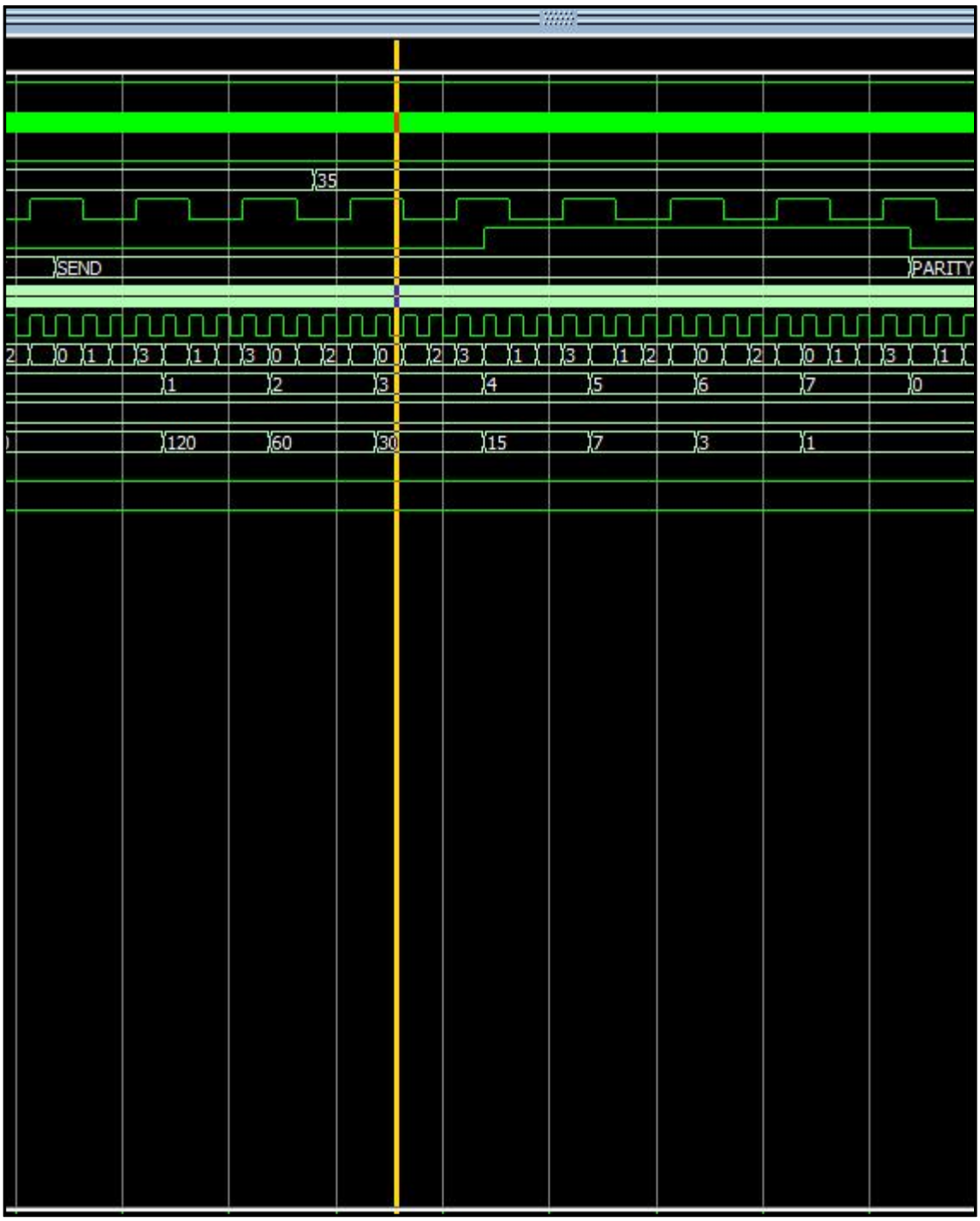


다음으로 START에서 SEND까지의 시뮬레이션 결과이다.

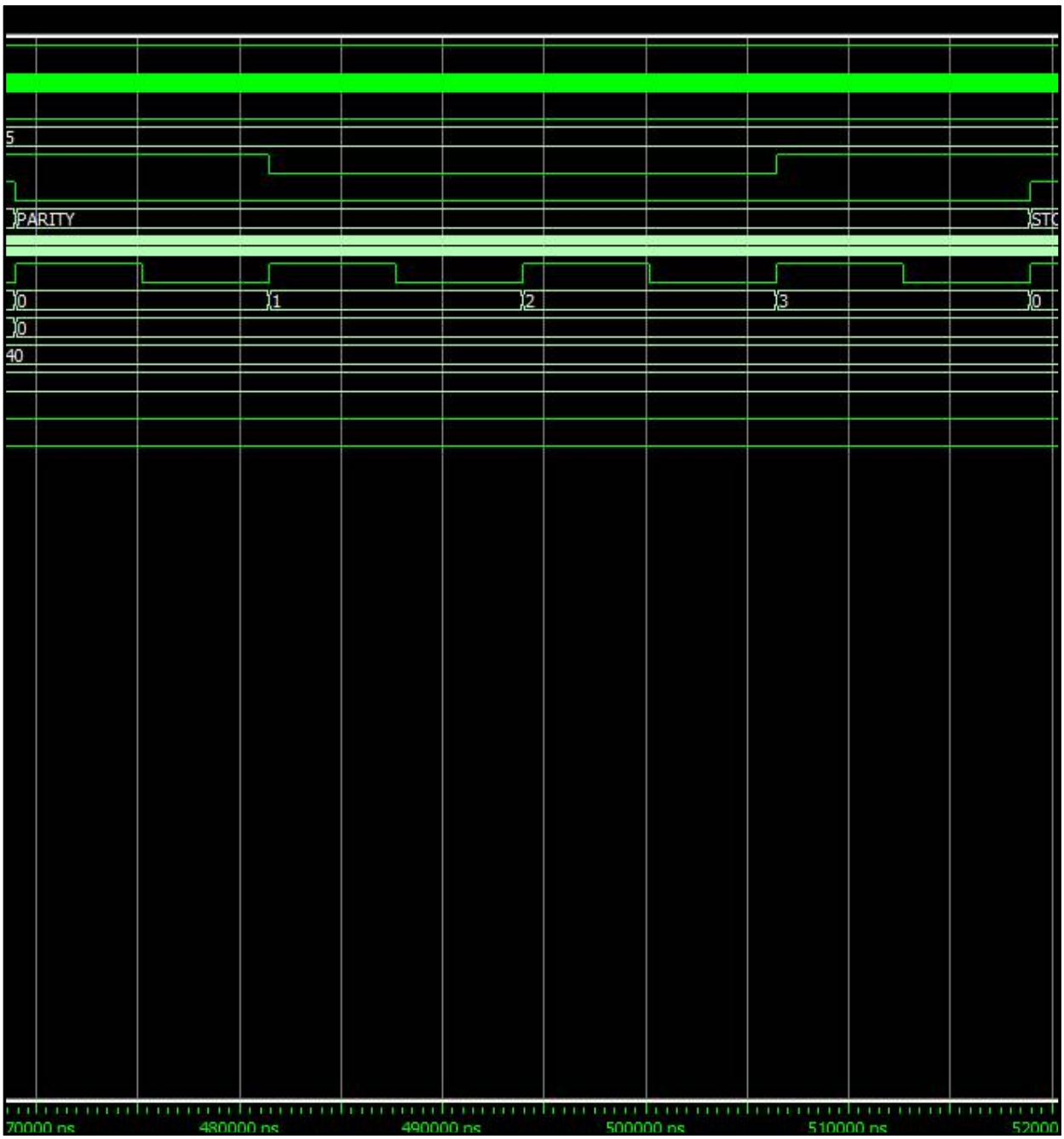
pclk_cnt의 값은 0, 1, 2, 3으로 pclk의 rising edge에 맞추어 증가함을 볼 수 있다.

그리고 pclk_cnt의 값이 '3'에 도달했을 시 state는 SEND로 이동하게 된다.

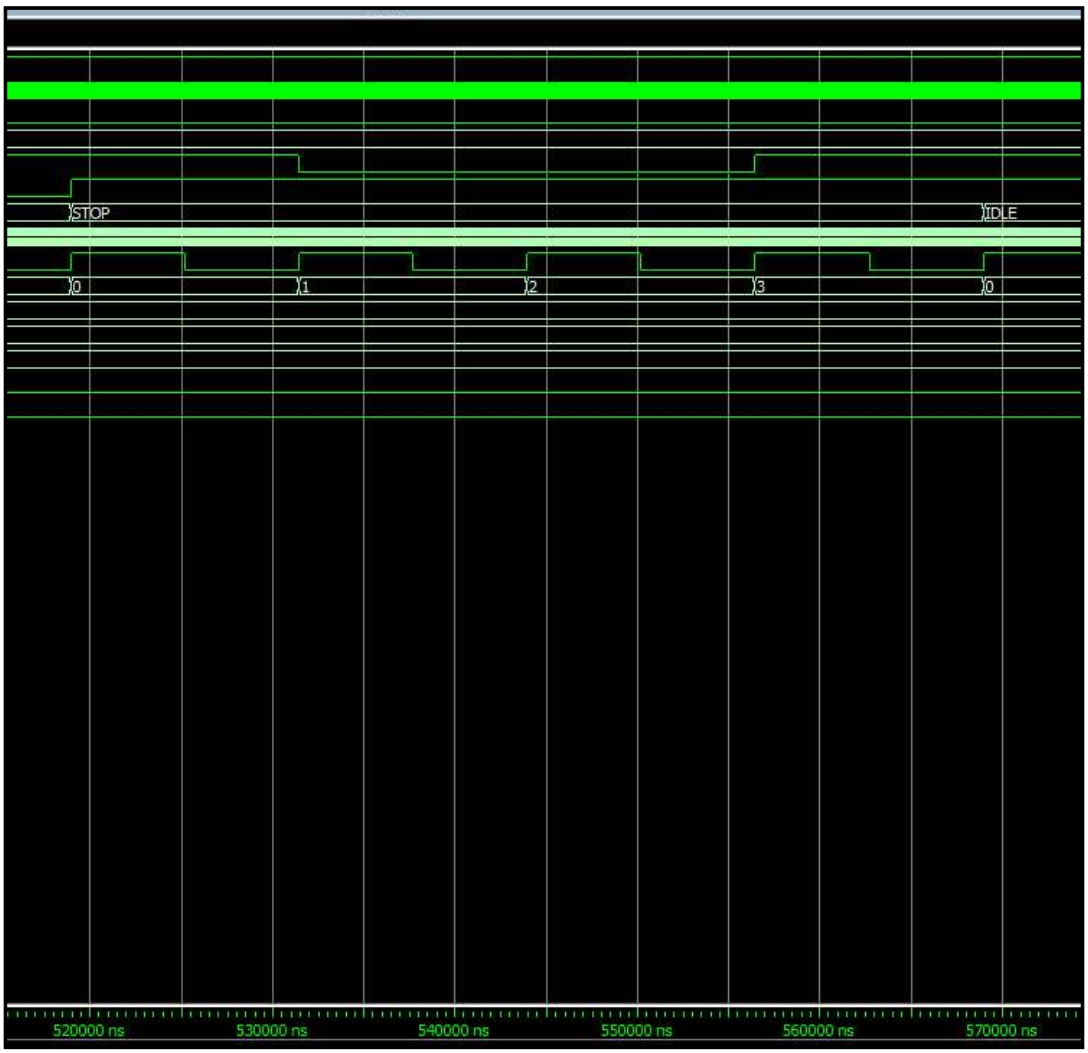
각 pclk_cnt의 값이 증가 할 때 마다(pclk의 rising edge이다) tx_data에 temp_data의 값을 입력받는데, 이 시뮬레이션에서 둘의 값은 같은 상태이기 때문에 움직임은 보이지 않는다.



SEND상태에서는 pclk_cnt의 값이 3이 될 때마다 bit_cnt의 값을 하나씩 증가시킨다.
그리고 tx_data의 값이 절반으로(반으로 나눈 후 정수 부분만을 취한다) 나뉘는 현상을 볼 수 있는데, 이는 위에 설명했듯
tx_data <= '0' & tx_data(7 downto 1) 의 코드가 있었기 때문이다.
tx_data는 오른쪽으로 shift되면서 0으로 원래 코드가 대체되는데, 이 때문에 값이 절반씩 줄어드는 것처럼 보이게 된 것이다.
이를 반복하다 bit_cnt이 값이 '7'에 도달하면 state는 PARITY로 이동하게 된다.

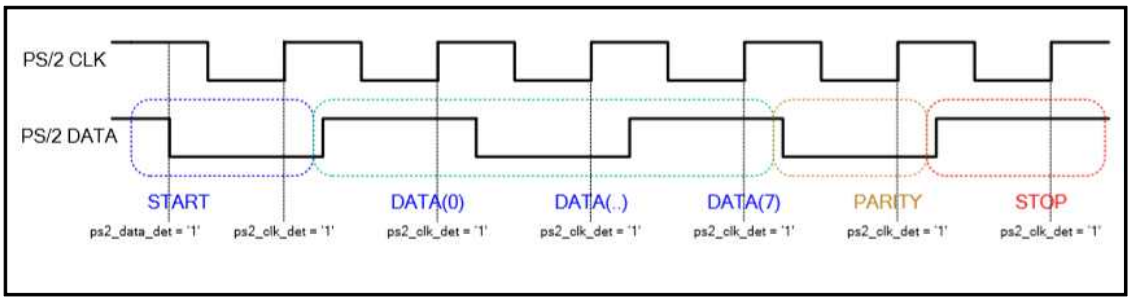


state가 PARITY 상태일 때
pclk_cnt가 '3'이 될 때까지 대기하가 '3'에 도달하면 STOP상태로 이동한다.

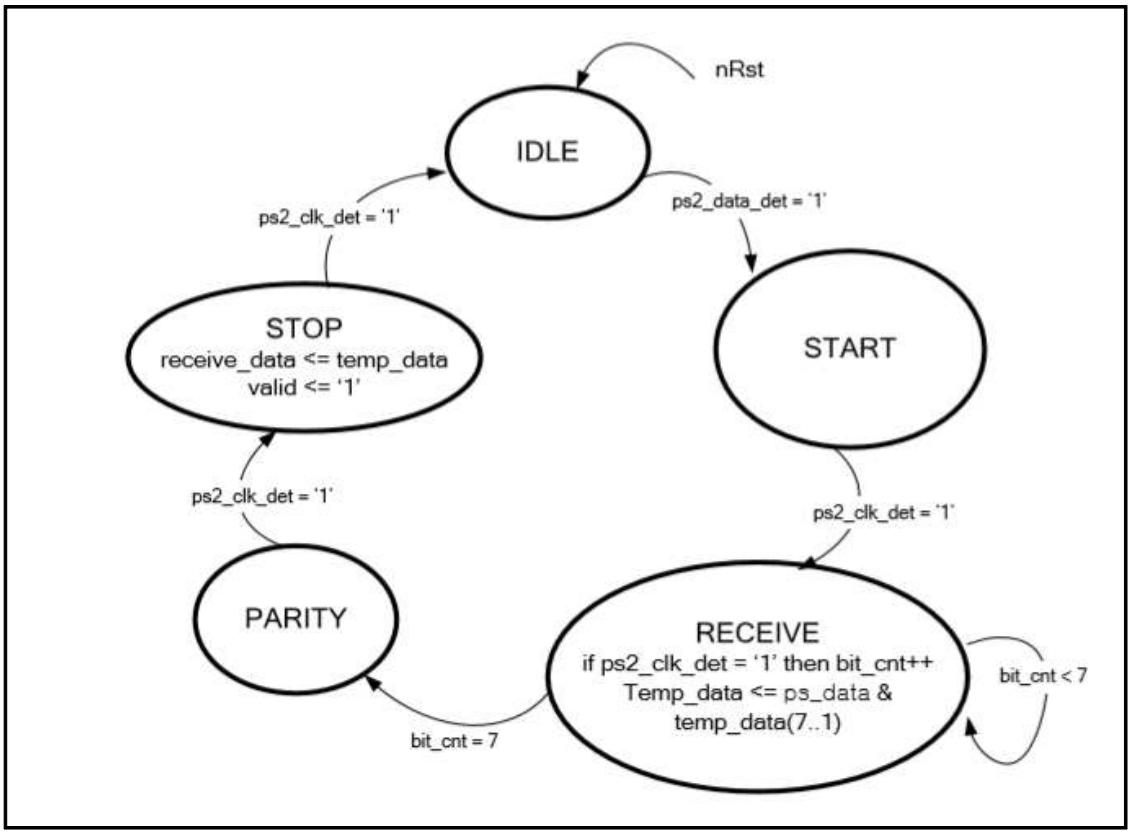


state가 STOP상태도 마찬가지로 pclk_cnt가 '3'일 때까지 대기하다 '3'에 도달하면 다시 IDLE상태로 이동하게 된다.

다음으로 PS2_Receiver을 설계한다.



먼저, PS/2 DATA가 falling edge일 때 START가 실행된다.
DATA와 PARTIY, STOP상태는 PS/2CLK의 falling edge상태에 맞춰 PS/2 DATA와 동기화 되어 진행된다.



nRst이 '0'인 상태에서 IDLE상태로 유지된다.
PS2/CLK가 falling edge인 상태에서 ps2_data_det가 '1'이면 IDLE에서 START로 이동한다.
다음부터는 PS2/CLK가 rising edge일 때 검사를 진행한다.
START에서 rising edge일 때 ps2_clk_det가 '1'이면 RECEIVE로 이동하여 RECEIVE에서 bit_cnt의 값이 7이 될 때까지 증가시킨다.
RECEIVE 상태에서는 ps2_clk_det가 '1'일 때 bit_cnt의 값을 증가시킨다.
이후 temp_data에 ps_data와 temp_data(7 downto 1)값의 and연산값을 저장한다.
bit_cnt의 값이 7인 상태에서 PARITY로 이동한다.
ps2_clk_det가 다시 '1'이 되면 STOP이 된다.
STOP상태에서 ps2_clk_det가 '1'이 되면 다시 IDLE상태로 돌아간다.

다음으로 PS2/receiver을 코딩한다

```
1
2   library ieee;
3     use ieee.std_logic_1164.all;
4     use ieee.std_logic_arith.all;
5     use ieee.std_logic_unsigned.all;
6
7   entity ps2_receiver is
8     port
9     (
10      nRst            : in std_logic;
11      clk             : in std_logic;
12      ps2_clk         : in std_logic;
13      ps2_data        : in std_logic;
14      received_data   : out std_logic_vector(7 downto 0);
15      valid           : out std_logic
16    );
17  end ps2_receiver;
```

위의 transmitter설명과 같다.

Library를 통해 연산 등의 단계에 필요한 로직을 불러내고, entity문에서 port를 디자인한다.

```
19  architecture BEH of ps2_receiver is
20
21    type state_type is(IDLE, START, RECEIVE, PARITY, STOP);
22    signal state        : state_type;
23    signal ps2_clk_d    : std_logic;
24    signal ps2_data_d   : std_logic;
25    signal ps2_clk_det  : std_logic;
26    signal ps2_data_det : std_logic;
27    signal bit_cnt      : std_logic_vector(3 downto 0);
28    signal temp_data    : std_logic_vector(7 downto 0);
```

Architecture를 진행하기 전, type과 signal을 각각 미리 설정한다.

각 단계에 대한 설명은 이전에 미리 했기에 생략한다.

```
30    begin
31
32      process(nRst, clk)
33        begin
34          if(nRst = '0') then
35            ps2_clk_d      <= '0';
36            ps2_clk_det    <= '0';
37            ps2_data_d     <= '0';
38            ps2_data_det   <= '0';
39          elsif rising_edge(clk) then
40            ps2_clk_d      <= ps2_clk;
41            ps2_data_d     <= ps2_data;
42
43            if(ps2_data_d = '1') and (ps2_data = '0') then
44              ps2_data_det <= '1';
45            else
46              ps2_data_det <= '0';
47          end if;
48
49          if(ps2_clk_d = '0') and(ps2_clk = '1') then
50            ps2_clk_det   <= '1';
51          else
52            ps2_clk_det   <= '0';
53          end if;
54        end if;
55      end process;
```

첫 번째 process문은 nRst, clk에 따라 동기화되어 진행된다.

nRst가 '0'일 때에 ps2_clk_d, ps2_clk_det, ps2_data_d, ps2_data_det의 값을 모두 '0'으로 초기화한다.

nRst이 '1'이고 clk가 rising_edge일 경우에 ps2_clk_d에 ps2_clk값을 넣고 ps2_data_d에 ps2_data를 입력한다.

- ➔ 그 상태에서 ps2_data_d가 '1'이고 ps2_data가 '0'이면 ps2_data_det에 '1'을 넣고
- ➔ 그렇지 않은 상태이면 ps2_data_det에 '0'을 넣는다.
- ➔ 이후 ps_clk_d 가 '0'이고, ps2_clk가 '1'인 상태라면 ps2_clk_det에 '1'을 입력하고
- ➔ 그렇지 않은 상태라면 ps2_clk_det에는 '0'을 넣는다

위의 4가지 상태에서 1, 2번째 줄과 3, 4번째 줄은 서로 영향을 주는 것을 알 수 있다.

```
57      process(nRst, clk)
58      begin
59          if(nRst = '0') then
60            state           <= IDLE;
61            bit_cnt         <= (others => '0');
62            temp_data       <= (others => '0');
63            received_data   <= (others => '0');
64            valid           <= '0';
```

다음 process에서

nRst이 '0'이면

state는 IDLE상태가 되고

모든 bit_cnt, temp_data, received_data, valid의 값을 '0'으로 초기화시킨다.

```
67          elsif rising_edge(clk) then
68          case state is
```

이후nRst이 '1'인 상태에서 clk가 rising_edge인 상태에서
state의 상태에 따라 각각의 상태가 진행된다.

```
70            when IDLE     =>
71              if(ps2_data_det = '1')then
72                state   <= START;
73              else
74                state   <= IDLE;
75              end if;
76                bit_cnt     <= (others => '0');
77                temp_data   <= (others => '0');
78                valid       <= '0';
```

이전 process에서 state는 IDLE상태에 위치할 것이라는 사실을 알 수 있다.
그 상태에서 ps2_data_det가 '1'이 되면 state는 START를 입력 받는다.
그렇지 않은 상태에서는 IDLE상태가 유지된다.
if문이 끝난 후에 bit_cnt, temp_data, valid의 모든 값을 '0'으로 초기화한다.

```
80            when START    =>
81              if(ps2_clk_det ='1') then
82                state     <= RECEIVE;
83              else
84                state     <= START;
85              end if;
```

state가 START의 상태에서
ps2_clk_det가 '1'이면 state는 RECEIVE로 진행된다.

```
87            when RECEIVE  =>
88              if(ps2_clk_det = '1') then
89                if(bit_cnt = 7) then
90                  bit_cnt   <= (others => '0');
91                  state     <= PARITY;
92                else
93                  bit_cnt   <= bit_cnt + 1;
94                  state     <= RECEIVE;
95                end if;
96                  temp_data <= ps2_data & temp_data(7 downto 1);
97                else
98                  state   <= RECEIVE;
99              end if;
```

state가 RECEIVE 상태에 도달하였을 때
ps2_clk_det가 1이면 bit_cnt의 값을 확인한다. (ps2_clk_det가 0이면 state는 RECEIVE상태에 머무른다.)
bit_cnt가 7이면 bit_cnt를 '0'으로 초기화하고 state는 PARITY로 진행한다.
bit_cnt가 7이 아닐 경우 bit_cnt에 1을 추가시킨다.
if문이 끝난 후에 temp_data에 ps2_data와 temp_data(7 downto 1)의 값을 연결 연산한다.

(위의 연결 연산의 결과는 시뮬레이션에서 설명하겠다.)

```
101           when PARITY    =>
102             if(ps2_clk_det ='1') then
103               state      <= STOP;
104             else
105               state      <= PARITY;
106             end if;
107             received_data    <= temp_data;
```

state가 PARITY값일 때
ps2_clk_det가 '1'이면 state는 STOP이 된다.
ps2_clk_det가 '1'이 아닐 경우 state는 PARITY에 머물게 된다.
이후 received_data에 temp_data의 값을 입력받는다.

```
109           when STOP      =>
110             if(ps2_clk_det ='1') then
111               state      <= IDLE;
112             else
113               state      <= STOP;
114             end if;
```

State가 STOP일 때
ps2_clk_det가 '1'이면 state는 IDLE로 되돌아간다.

```
116           when others    =>
117             state <= IDLE;
118           end case;
119         end if;
120       end process;
121
122
123
124   end BEH;
```

state가 위의 when상태에 해당하지 않을 경우 IDLE상태로 보낸다.
위의 코딩을 마치고 BEH를 마친다.

이어서 PS2/receiver의 Test bench를 코딩한다.

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_arith.all;
4   use ieee.std_logic_unsigned.all;
5
6   entity tb_ps2_receiver is end;
7
8   architecture BEH of tb_ps2_receiver is
9
10    component ps2_keyboard is
11      port
12      (
13        nRst          : in std_logic;
14        clk           : in std_logic;   -- 100MHz, 10ns
15        start_sig     : in std_logic;
16        data          : in std_logic_vector(7 downto 0);
17        ps2_clk       : out std_logic;
18        ps2_data      : out std_logic
19      );
20    end component;
21
22    component ps2_receiver is
23      port
24      (
25        nRst          : in std_logic;
26        clk           : in std_logic;
27        ps2_clk       : in std_logic;
28        ps2_data      : in std_logic;
29        valid         : out std_logic;
30        received_data : out std_logic_vector(7 downto 0)
31      );
32  end component;
33
34  signal nRst         : std_logic;
35  signal clk          : std_logic;
36  signal start_sig    : std_logic;
37  signal data         : std_logic_vector(7 downto 0);
38  signal ps2_clk      : std_logic;
39  signal ps2_data     : std_logic;
40  signal valid        : std_logic;
41  signal received_data: std_logic_vector(7 downto 0);
42  signal internal_cnt : std_logic_vector(80 downto 0);
```

위의 transmitter의 T_B와 마찬가지로 entity를 선언한 뒤 signal과 각 포트를 design한다.

```
44  begin
45
46    process
47      begin
48        if(NOW = 0 ns) then
49          nRst <= '0', '1' after 200ns;
50        end if;
51        wait for 1sec;
52      end process;
53
54    process
55      begin
56        clk <= '0', '1' after 5ns;
57        wait for 10ns;
58    end process;
59
60    process(nRst, clk)
61      begin
62        if(nRst = '0') then
63          internal_cnt <= (others => '0');
64        elsif rising_edge(clk) then
65          internal_cnt <= internal_cnt + 1;
66        end if;
67      end process;
```

세 process를 순서대로 설명하면
nRst에 처음 0~199ns까지는 '0'을, 200ns~1s까지는 '1'을 할당한다.

clk에 '0'을 입력받다가 5ns~10ns까지는 '1'을 할당하여 10ns짜리의 50% PWM을 만든다.
(이는 clock신호로 활용된다.)

nRst가 '0'일 때 internal cnt의 모든 bit를 '0'으로 초기화한다.
nRst이 '0'이 아니고, clk신호가 rising edge인 상황에서 internal_cnt는 '1'씩 증가한다.

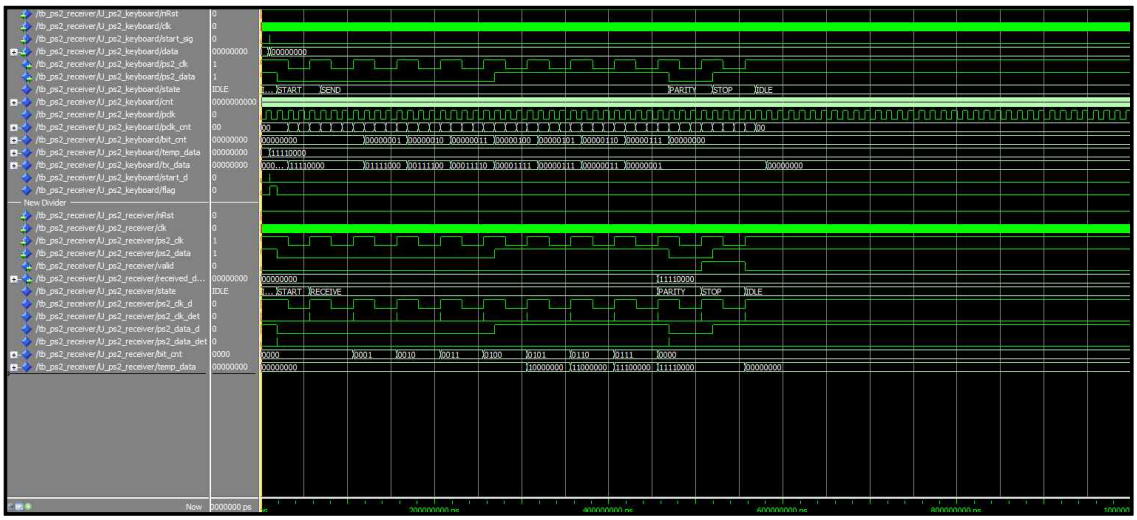
```
69      start_sig <= '1'   when internal_cnt  = 1000   or  internal_cnt  = 200000
70      data      <= x"F0" when internal_cnt >= 900    and internal_cnt <= 1100
71                   x"23" when internal_cnt >= 190000 and internal_cnt <= 210000
72                   (others => '0');
```

start_sig는 internal_cnt가 1000 or 200000의 값을 가질 때에만 '1', 그렇지 않을 때는 항상 '0'의 값을 갖는다.
internal_cnt가 900~1100의 값을 때에 data는 (0xf0)의 값을, 190000~210000의 값을 때에는 (0x23)의 값을 할당받는다.

```
74      U_ps2_keyboard : ps2_keyboard
75      port map
76        (
77          nRst          => nRst,
78          clk           => clk,
79          start_sig     => start_sig,
80          data          => data,
81          ps2_clk       => ps2_clk,
82          ps2_data      => ps2_data
83        );
84
85      U_ps2_receiver : ps2_receiver
86      port map
87        (
88          nRst          => nRst,
89          clk           => clk,
90          ps2_clk       => ps2_clk,
91          ps2_data      => ps2_data,
92          valid         => valid,
93          received_data => received_data
94        );
95
96      end BEH;
```

각 포트를 할당한 후 test_bench의 코딩을 마친다.

두 receiver의 keyboard와 Test_bench의 시뮬레이션을 진행한다.

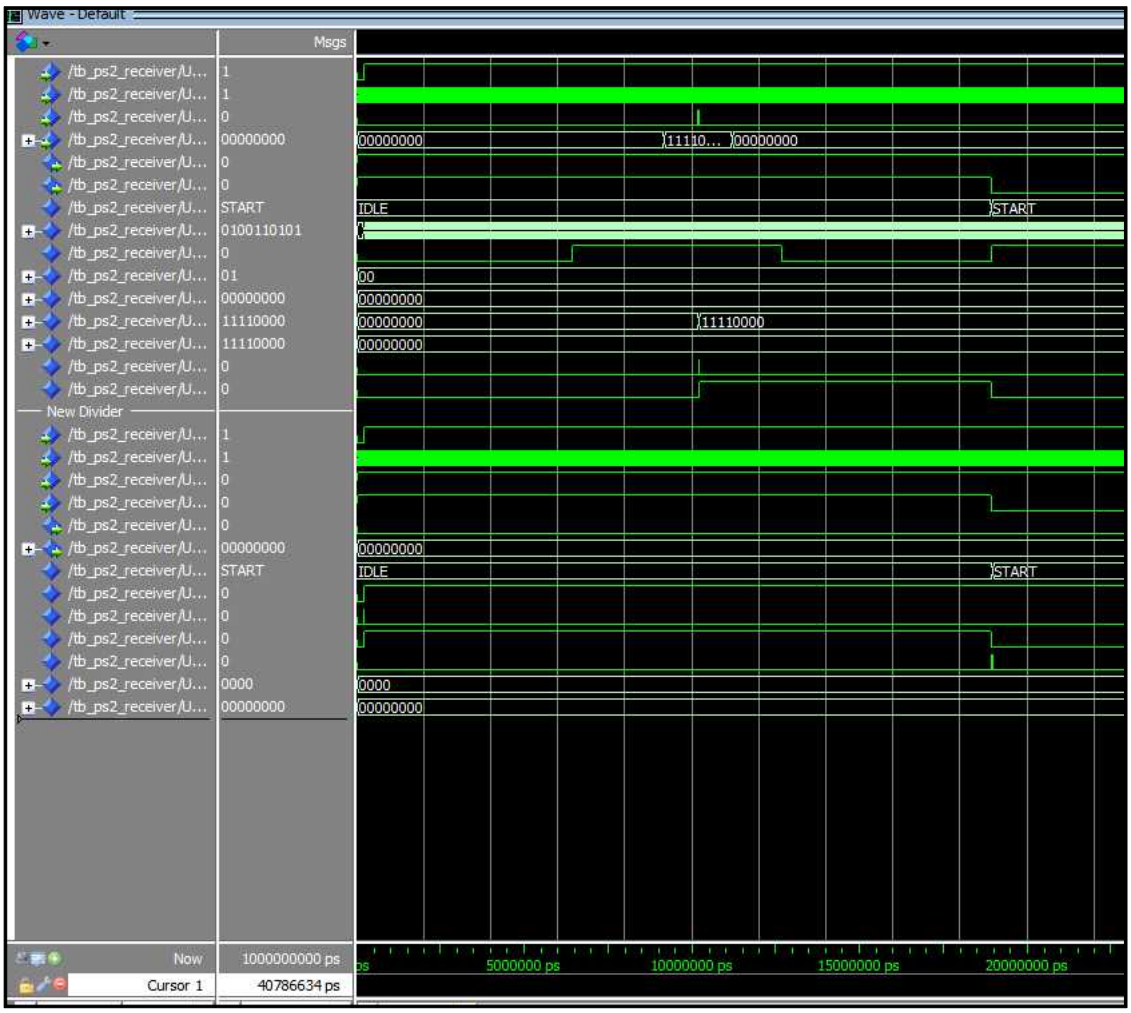


두 시뮬레이션은 DIveder에 의하여 나뉜 상태인데, 알아보기 쉽게 transmitter와 receiver을 나누었다. 이는 divice의 port명을 보면 확인할 수 있다.

처음에 nRst의 값이 '0'인 상태에서 state는 IDLE상태이다. nRst은 200ns 뒤에 '1'을 부여받는다.

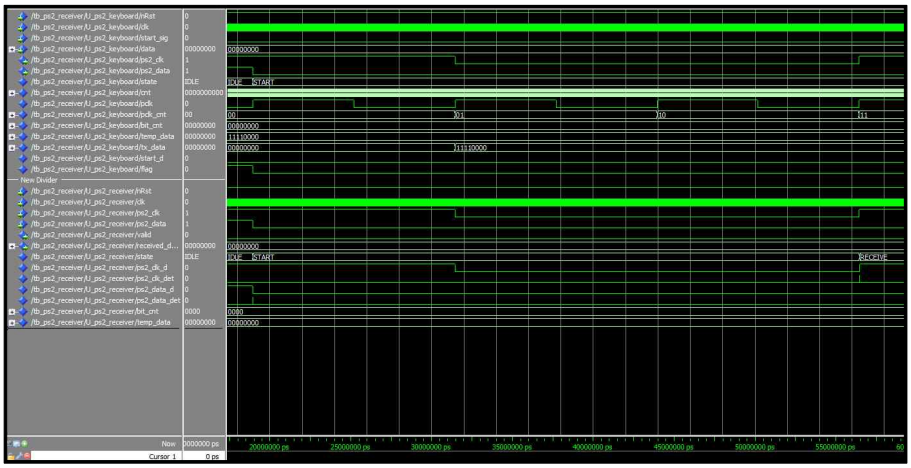
nRst의 값이 '0'인 상태에서bit_cnt, temp_data, received_data, valid는 모두 '0'으로 초기화된 상태이다.

이후 clk가 rising edge일 때에 상태가 변화한다.

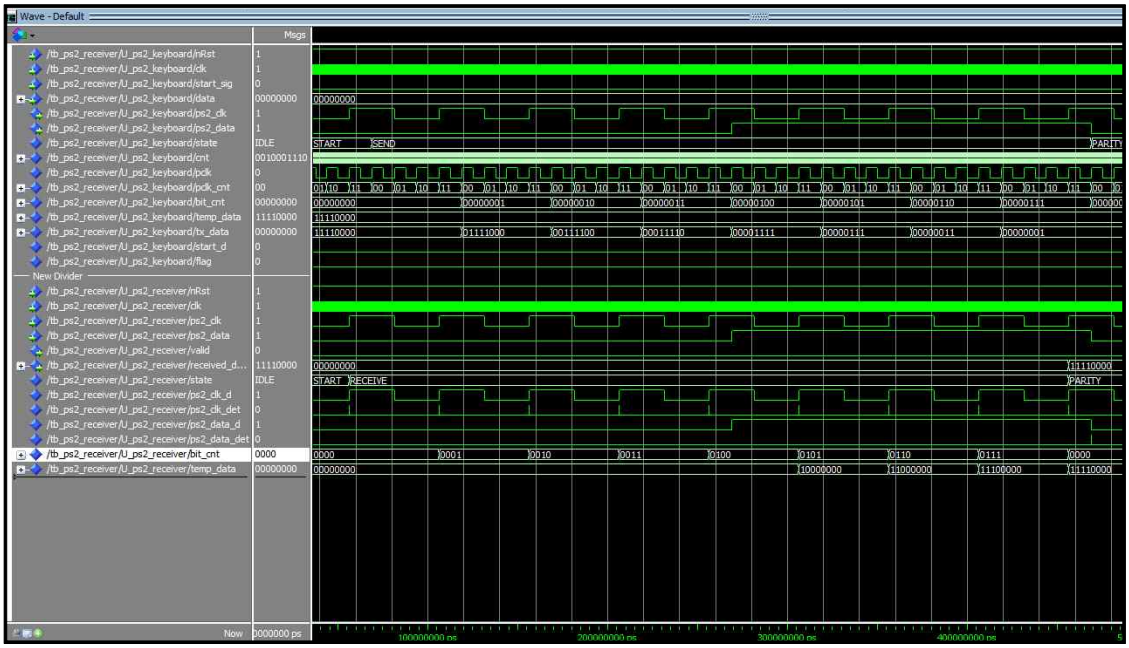


먼저 IDLE상태에서 ps2_data_det가 '0'일 때에는 state가 유지되다가, ps2_data_det가 '1'이 되면 state는 START로 넘어가게 된다.

그리고 bit_cnt, temp_data, valid의 모든 값을 '0'으로 초기화한다.



이전과 마찬가지로 ps2_clk_det가 '1'이 되면 state는 RECEIVE로 넘어간다.

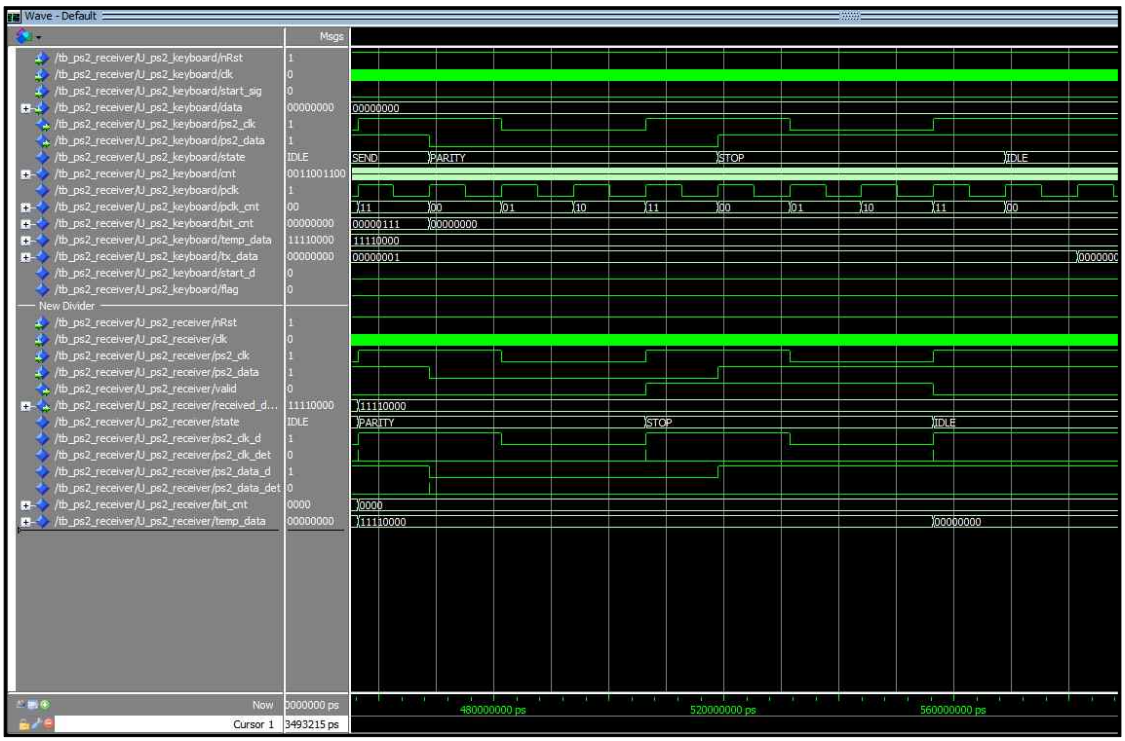


RECEIVE에서 PARITY로 진행되기 전까지 ps2_clk_det가 '1'일 때마다 bit_cnt는 1씩 증가한다.

또한 temp_data의 7~1의 비트와 ps2_data를 '&'연산 즉 연결 연산하여 그 값을 temp_data에 넣는다(여기서 본래 temp_data와 ps2_data는 0이여서 변경이 없다가, temp_data가 1이 된 순간부터 temp_data는 1씩 증가하며 shift됨을 확인 가능하다)

반복하다 bit_cnt가 7이 되면 다시 0으로 초기화 된 후 PARITY 상태가 된다.

이후 received_data에 temp_data 값을 저장시킨다.



다음으로 ps2_clk_det가 '1'이 될 때 마다 state는 PARITY -> STOP -> IDLE로 이동한다.

+> state가 STOP상태일 때 valid는 '1'의 값을 할당받는다.

이를 통해 PS2/Transmitter와 Receiver을 구현하였다.

이를 키보드에 장입시킨 후 확인해 보면

Ex) d : 35 = 0011,0101 ----→ ps2_data(10101100)

d : F0 = 1111,0000 -----→ ps2_data(d0 d1 d2 d3 d4 d5 d6 d7)

이와 같이 transmitter와 receiver에 나타남을 알 수 있다.

Report 소감.. (결과 및 고찰)

먼저 처음 report를 적으면서 어떻게 진행해야 하는지, 어떤 방식으로 코딩과 시뮬레이션을 진행해야 하는지에 대해서 고민이 많았다.

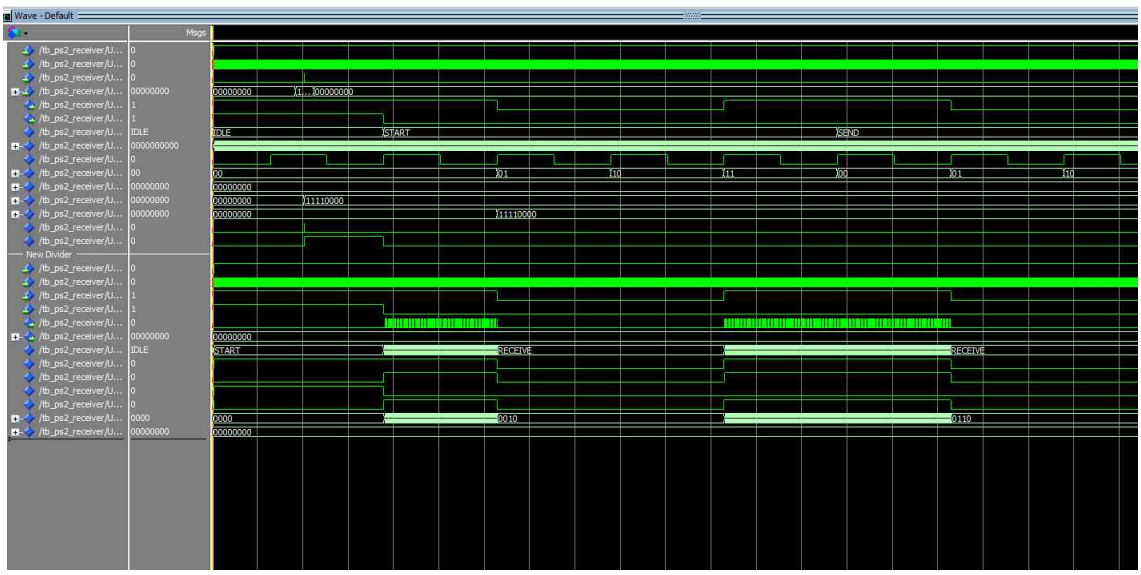
학교에서 Test_Bench에 대해 처음 배운 주에 예비군을 갔었기 때문에 듣지 못했고, 이후로는 바로 실습이 진행되어 따라가기에 많이 힘에 부쳤던 때문이다.

때문에 주변 동기들과 먼저 수업을 이수한 선배들에게 Test_bench를 이용한 시뮬레이션 방법에 대해 공부를 하였고 그 이후 레포트를 작성하자는 생각을 했다.

그런데 오히려 그렇게 진행을 했더니 더 수월하게 공부 할 수 있었던 것 같은 느낌이다.

학교에서 수업을 녹음하며 이해하지 못한 상태로 필기하고, 코딩했던 부분을 밤을 새가며 다른 학생들에게 배운 후에 다시 공부하였더니 더 재미있고 빠르게 습득되었다.

그를 통해 시뮬레이션과 코딩을 하였는데, 잘못 코딩한 부분이 오류가 났었다.
작은 오타였는데, 그것 때문에 RECEIVER에서 생각지 못한 시뮬레이션이 발생했다.



재미있게도 이 오류 덕분에 SOC에 대해 더 깊이 이해할 수 있게 되었다.

당시까지만 해도 마우리 찾아도 어디가 잘못된지 알 수 없어, 시뮬레이션의 오류 이유를 찾기 위해 모든 PORT를 매칭하며 잘못된 부분을 찾아 보았다.

그렇게 처음부터 모든 코드와 시뮬레이션을 비교하며 동작의 원리와 결과를 정독하다보니 keyboard receiver의 첫 번째 process문에서 ps2_clk신호와 ps2_data신호를 반대로 적었음을 알아차리게 되었다.

그 후 잘못된 부분을 수정하고 다시 코딩을 하자 위의 시뮬레이션처럼 문제 없이 실습을 마칠 수 있었다.

이후로 이해가 잘 가지 않았던 부분은 '&' 연산자에 대한 부분인데, 방학동안 공부했던 JAVA언어나 다른 컴퓨터 언어에서 &연산자는 'and'로, 보통 2진수에서는 이전의 숫자에 곱하는데 여기서는 다른 방식이라 조금 헷갈렸다.

그것을 해결하기 위해 인터넷 검색과 자습을 바탕으로 많은 공부를 하였고, 실습을 마칠 때에는 모든 코딩을 직접 짜는건 힘들어도 이해할 수는 있게 되었다.

아무것도 모르는 상태에서 실습을 진행할 때는 막막하고, 오류가 발생할 때에는 화가 나기도 했지만 이번 실습을 했기 때문에 적어도 PS2 keyboard의 코딩에 필요한 모든 요소와 PORT, 시뮬레이션을 위한 Test_bench에 대해 많은 공부를 하였고, 이후의 수업에도 도움이 될 지식을 쌓을 수 있었다.

그리고 다음 수업부터는 진도를 빠르게 따라잡을 수 있어 다른 이들의 도움 없이도 충분히 혼자 진행 할 수 있을 것이라 생각한다.